VŠB – Technická univerzita Ostrava

Fakulta elektrotechniky a informatiky

Katedra informatiky

Redukce sítí a jejich vizualizace

Reduction and visualization of meshes

2009 Zdeněk Neustupa

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 7. května 2009 .....

Na tomto místě bych velice rád poděkoval vedoucímu své bakalářské práce
Ing. Petru Gajdošovi, Ph.D. za veškerou pomoc a podporu, kterou mi poskytl.

Abstrakt

Tradiční metody pro získávání dat popisujících 3D modely většinou produkují velice podrobné a husté sítě. Přestože tato robustnost dat může být žádoucí pro korektní matematické výpočty, pro rychlou a paměťově málo náročnou vizualizaci to neplatí. Z tohoto důvodu je nutné daná data jistým způsobem zjednodušit tak, aby byla vizuální podoba redukovaných dat co nejbližší datům původním a přitom se dalo s redukovanými modely pracovat v relativně krátkém čase.

Cílem této práce je podat přehled existujících řešení v oblasti redukce 3D sítí. Navrhnout a implementovat jednoduchou aplikaci, která bude implementovat určitou redukční metodu. Dále porovnat výsledky redukce naší aplikace se zvolenou metodou a v neposlední řadě pak vizualizovat výsledky v openGL.

Klíčová slova: polygonální síť, trojúhelníková síť, decimace vrcholů, zhroucení hrany, shlukování vrcholů, redukce, progresivní reprezentace

Abstract

Traditional data gaining methods for 3D models description usually produces large and high detailed meshes. Although this data robustness might be desirable for correct mathematical calculations, it is not desirable for fast and low-memory visualization. For this reason it is necessary to reduce this data while preserving a good visual approximation to the original data. Also we need to be able to work with the reduced model in a relatively brief time.

The goal of this thesis is to describe existent solutions in 3D mesh reduction problematic and to design and implement simple application for mesh reduction method. Further to contrast the results of the reduction method of our application and the chosen one and not least to visualize these results using openGL.

Keywords: polygonal mesh, triangular mesh, vertex decimation, edge collapse, vertex clustering, simplification, progressive representation

Seznam použitých zkratek a symbolů

3D	–	trojrozměrný
PM	–	progresivní reprezentace sítě (progressive mesh)
CLR	–	Common Language Runtime (implementace CLI firmou Microsoft)
CLI	–	Common Language Infrastructure (prostředí pro vykonávání kódu)
CIL	–	Common Intermediate Language (byte kód pro spuštění CLR)

Obsah

Seznam tabulek

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

1.1 Úvod do problematiky

Jelikož samotná oblast počítačové grafiky zaznamenala v posledních 10 letech veliký posun vpřed, musel být učiněn i velký pokrok v možnostech kvalitního a rychlého zobrazování. Jak už bylo naznačeno, objemnost dat, která je vhodná pro korektní a přesné matematické výpočty nad polygonální sítí, je pro její rychlou a paměťově málo náročnou vizualizaci problémem. Takováto data je nutno smysluplně redukovat, aby se s nimi dalo rychle a efektivně manipulovat a přitom byla zachována vizuální podoba s daty původními. Tímto úkolem se zabývá oblast redukce 3D sítí (3D mesh).

Oblast redukce 3D sítí se zabývá nejen samotnou redukcí těchto modelů, ale také řešením problémů vyvstávajících při samotném získávání těchto modelů pomocí standardních procedur (3D skener, atd.). Tradiční metody získávání dat popisujících modely pomocí planárních plošek, jako jsou například trojúhelníky, velmi často produkují zbytečně podrobné sítě s navíc neúměrným počtem trojúhelníků v závislosti na zakřivení dané plochy. Tím pádem se na téměř rovinné ploše může vyskytovat stejný počet trojúhelníků jako na ploše s maximálním zakřivením. Takovéto trojúhelníky mají velice malý vliv na výslednou podobu sítě. Úkolem redukčních metod je takovéto trojúhelníky a jim podobné odstranit a nahradit je menším počtem trojúhelníků, při zachování co největší podobnosti s původní sítí.

Tato práce je zaměřena na právě takovéto redukční metody se zaměřením na redukci trojúhelníkových sítí.

1.2 Rozdělení dokumentu

Na úvod této práce bude podán přehled již existujících metod redukce trojúhelníkových sítí, které jsou významné pro tuto problematiku. Jelikož existuje mnoho redukčních metod založených na velice odlišných principech, byly vybrány metody, z nichž

žádné dvě nepatří do téže skupiny principu redukce. Na závěr 2. kapitoly budou shrnuty všechny parametry metod v ní popsaných a bude z nich vybrána jedna, kterou se pokusíme implementovat. Implementovaná metoda může být mírně pozměněna.

V další části této práce bude popsán návrh aplikace včetně použitých technologií a návrhu datových struktur, které budeme implementovat pro realizaci redukčního algoritmu.

V následující kapitole bude blíže popsán námi vybraný redukční algoritmus a s ním související triangulační postupy.

Poté porovnáme výsledky redukce naší aplikace s danou metodou a budeme tyto výsledky vizualizovat pomocí $openGL$.

Na závěr shrneme dosažené výsledky a přínosy této práce k danému tématu.

2 Přehled redukčních metod

Během několika posledních let byl v oblasti redukce sítí, vývoje datových struktur s velkým počtem rozlišení a editace sítí učiněn velký pokrok. Během této velice plodné doby byl vydán nespočet odborných článků, které prezentovaly nové metody pro redukci polygonálních sítí, či návrh datových struktur, které byly schopny zachytit větší počet rozlišení těchto sítí. Úkolem této práce není prezentovat veškeré metody, které byly za posledních dvacet let vyvinuty. V následujících kapitolách budou popsány důležité redukční metody či návrhy datových struktur, které se významně zapsaly do dějin problematiky redukce polygonálních sítí.

2.1 Shlukování vrcholů [1]

Algoritmus založený na shlukování vrcholů prezentovali pánové Dmitry Brodsky a Benjamin Watson v roce 2000 pod názvem R-simp algoritmus [1]. Oproti jiným algoritmům R-simp algoritmus začíná hrubou aproximací modelu a zjemňuje ji, dokud není dosaženo požadované složitosti modelu. Algoritmus začíná pracovat s modelem v jednom jediném clusteru (cluster je kolekce trojúhelníků z původního modelu). Počáteční cluster je poté rozdělen na osm sub-clusterů. Tyto sub-clustery jsou dále iterativně děleny, dokud není dosažen potřebný počet clusterů (vrcholů). Clustery jsou pro dělení vybírány podle velikosti odchylky normálového zakřivení povrchu daného clusteru. R-simp algoritmus může být rozdělen na tři fáze:

Inicializace: V této fázi se vytvoří tzv. souhrnný seznam trojúhelníků (gfl) a vrcholů (gvl), stejně jako spojovací seznamy vrchol-vrchol a vrchol-trojúhelník. Také je vytvořeno osm počátečních clusterů.

Simplifikace: V této fázi dochází ke zjednodušování modelu. Zjednodušovací proces sestává ze čtyř kroků:

1. výběr clusteru, který má nejvyšší hodnotu normálové odchylky

2. rozdělení tohoto clusteru na základě velikosti a směru normálové odchylky povrchu
3. výpočet normálové odchylky povrchu pro každý sub-cluster
4. opakovat dokud není dosaženo požadovaného počtu clusterů (vrcholů)

Post zpracování: Pro každý zbylý cluster vypočítat zastupující vrchol (centroid). Znovu triangulovat model.

2.1.1 Inicializace

Během inicializační fáze jsou vytvořeny globální seznamy vrcholů a trojúhelníků a osm počátečních clusterů. Počáteční clustery jsou určeny rozdělením modelu podle tří rovin zarovnaných podle os, které jsou umístěny do středu výřezového kvádru modelu. Poté se vypočítá velikost normálové odchylky každého clusteru. Těchto osm clusterů je poté vloženo do prioritní fronty, uspořádané podle velikosti normálové odchylky.

2.1.2 Výběr clusteru pro dělení

V simplifikační fázi algoritmu je prvním krokem výběr clusteru, ve kterém se normály liší nejvíce (cluster na začátku fronty). Spočítáme velikost normálové odchylky za použití obsahem vyváženého průměru $(\vec{mn})$ všech normál trojúhelníků daného clusteru.

Čím více je povrch rovinný, tím je větší velikost $(\vec{mn})$. Pokud jsou všechny trojúhelníky koplanární, velikost $(\vec{mn})$ je rovna obsahu povrchu clusteru. Tuto složku, naší míry normálové odchylky plochy, definujeme následovně:

$$cp = \frac{\|\vec{mn}\|}{\sum_i^N a_i} \tag{1}$$

2.1.3 Dělení clusteru

Rozdělení clusteru se skládá ze čtyř kroků. Nejdříve je nutno určit kolik rovin bude potřeba pro rozdělení clusteru. Poté orientovat tyto roviny. A nakonec je umístit a vytvořit nový sub-cluster.

Cluster je rozdělen na dva, čtyři či osm sub-clusterů, jejichž počet je závislý na velikosti zakřivení plochy. Nechť c_{mn}, c_M a c_m jsou rovny vlastním hodnotám, které představují velikost základního zakřivení. Nechť $\vec{c}_M$ a $\vec{c}_m$ představují odpovídající vlastní vektory (ty jsou ve vztahu ke směru maximálního a minimálního zakřivení).

Pokud jsou všechny vlastní hodnoty podobné velikosti, charakter normálového zakřivení je nejasný. Proto následující dvě podmínky porovnávání vlastních hodnot musí být pravdivé: $c_M > 2c_m$ a $c_{mn} < 2c_M$. V takovémto případě rozdělíme cluster na osm sub-clusterů. První dělící rovina je kolmá k $\vec{c}_M$, druhá rovina je kolmá k $\vec{c}_m$ a třetí rovina je kolmá k $\vec{mn}$.

Pokud je $\frac{c_M}{c_m} \leq 4$, pak je povrch s největší pravděpodobností polokulový, jelikož je zde významné zakřivení jak v minimálním tak maximálním směru zakřivenosti. V takovémto případě dělíme cluster na čtyři sub-clustery. První dělící rovina je kolmá k $\vec{c}_M$ a druhá rovina je kolmá k $\vec{c}_m$.

Ve zbylých případech, kdy je $\frac{c_M}{c_m} > 4$, je plocha s největší pravděpodobností válcová, jelikož nejvíce zakřivení je pouze v jednom směru. V tomto případě dělíme cluster na dva sub-clustery. Dělící rovina je kolmá k $\vec{c}_M$.

Sub-clustery vznikají rozdělením vrcholů do clusterů. Přiřazení vrcholu k jednotlivým clusterům, závisí na poloze daného vrcholu vůči dělícím rovinám. Při rozdělení do clusterů trojúhelníky následují vrcholy. Trojúhelník může patřit do dvou či tří clusterů, pokud vrcholy tohoto trojúhelníku padnou do různých sub-clusterů.

I pokud je celý model topologicky propojen, může se stát, že daný cluster bude obsahovat dvě či více oddělených součástí. Spojení takovýchto částí jediným vrcholem

může vést k závažné deformaci. Zdá se tedy velice užitečné provést kontrolu topologie, abychom zjistili zda nový cluster obsahuje oddělené části.

Pokud cluster obsahuje nepropojené části, každá část je pak umístěna do samostatného clusteru. I přesto, že kontrola topologie prodlužuje celkový čas redukce, je nárůst kvality redukce značný.

2.1.4 Post zpracování

Jakmile je ukončena část simplifikace, je nutno udělat ještě dvě věci. První je spočítat zastupující vrchol pro každý cluster. Druhá je re-triangulovat výstupní povrch.

Pro co nejpřesnější reprezentaci daného clusteru pomocí vrcholu je nutné, aby byl vrchol co nejblíže všem trojúhelníkům v clusteru. Spočítáme všechny minimalizované sumy vzdáleností od rovin obsahující trojúhelníky clusteru. A minimalizujeme vzdálenost na druhou.

Poté co je spočítán zastupující vrchol pro každý cluster, vrcholy jsou vloženy do zjednodušeného seznamu vrcholů (svl). Všechny vrcholy v gvl, jejichž odkazy jsou obsaženy v seznamu vrcholů daného clusteru, jsou také přidány do nového seznamu vrcholů (svl). Poté projdeme globální seznam trojúhelníků. Jakýkoliv trojúhelník odkazující se na tři různé vrcholy v svl je udržován a přidán do sfl. Všechny ostatní trojúhelníky degenerují na hrany či vrcholy a jsou odstraněny.

2.2 Decimace vrcholů [2]

Decimace vrcholů je jednou z nejstarších metod redukce trojúhelníkových sítí. Jakožto každá jiná redukční metoda si dává za úkol snížit celkový počet trojúhelníků v síti, při zachování původní topologie a dobré aproximace k síti původní. Metoda se také nazývá Schroederova metoda, jelikož byla poprvé prezentována Williamem J. Schroederem a dalšími autory v roce 1992 [2].

2.2.1 Popis metody

Samotný redukční algoritmus je relativně jednoduchý. Přes všechny vrcholy sítě jsou prováděny několikanásobné průchody. Při průchodu je každý vrchol kandidátem na odstranění, a pokud se shoduje s určitými stanovenými redukčními kritérii, je vrchol smazán a všechny trojúhelníky, ve kterých daný vrchol ležel, jsou taktéž odstraněny. Výsledná díra je následně zacelena lokální triangulací. Odstraňování vrcholů pokračuje, s možným přednastavením redukčních kritérií, tak dlouho, dokud není splněna určitá ukončovací podmínka. Obvykle je ukončovacím kritériem stanovena určitá procentuální redukce sítě, či maximální redukční hodnota. Algoritmus je možno rozdělit na tři části:

1. charakterizace lokální geometrie a topologie vrcholu
2. ohodnocení dle redukčního kriteria
3. triangulace vzniklé díry

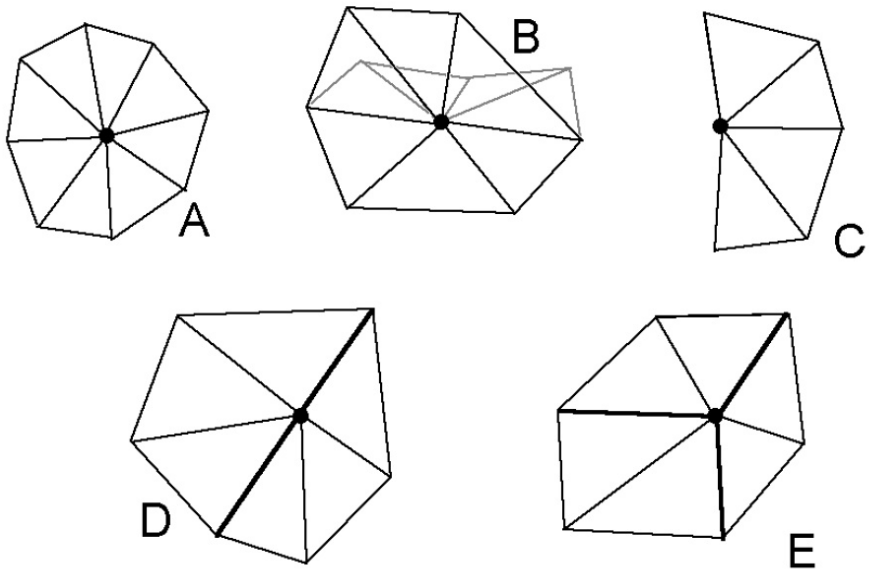
2.2.2 Charakterizace lokální geometrie/topologie

Prvním krokem redukčního algoritmu je charakterizovat lokální geometrii a topologii pro daný vrchol. Výsledek tohoto procesu určí, zda-li je daný vrchol kandidátem na odstranění, a pokud ano, jaká kritéria se mají použít.

Každý vrchol může být zařazen do jedné z pěti možných kategorií: jednoduchý, složitý, hraniční, na vnitřní hraně či rohový. Příklad každé kategorie je ukázán na obrázku 1.

Jednoduchý vrchol (obr. 1, A) je obklopen úplným cyklem trojúhelníků a každá hrana vycházející z tohoto vrcholu je incidentní právě se dvěma trojúhelníky.

Pokud je hrana incidentní s více jak dvěma trojúhelníky, nebo vrchol leží v trojúhelníku, který není v daném cyklu, jedná se o složitý vrchol (obr. 1, B). Toto jsou $non - manifold$ případy.



Obrázek 1: Pět možných klasifikací vrcholů, Zdroj: [5]

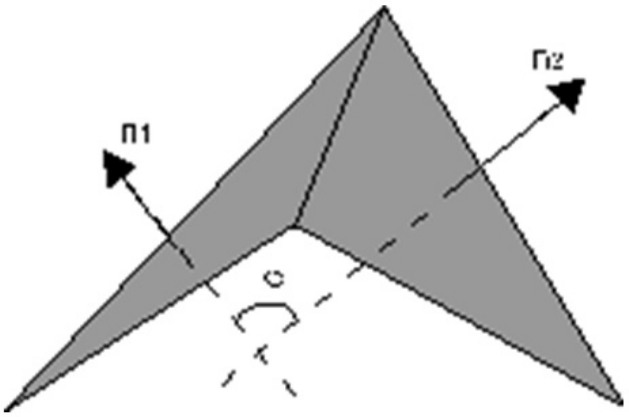
Vrchol, který je na okraji sítě, například v půl cyklu trojúhelníků, je nazýván vrchol hraniční (obr. 1, C).

Pokud je prostorový úhel mezi dvěma sousedními trojúhelníky větší než určitý mezní úhel, jedná se o hranu důležitou. Prostorový úhel mezi dvěma rovinami je určen jako úhel, který svírají jejich normály. Tato situace je naznačena na obrázku 2. Pokud z vrcholu vychází dvě důležité hrany, jedná se o vrchol na vnitřní hraně (obr. 1, D), pokud z něho vychází jedna nebo tři a více důležitých hran, jedná se o vrchol rohový (obr. 1, E).

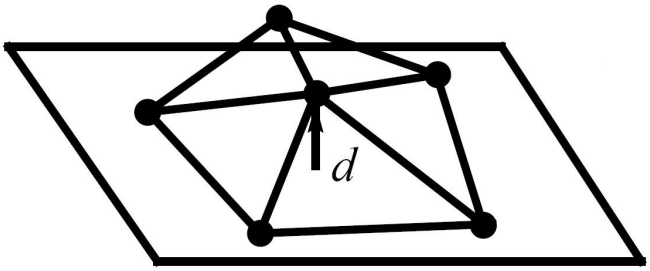
Složité vrcholy se ze sítě neodstraňují. Všechny ostatní vrcholy jsou kandidáty na odstranění.

2.2.3 Ohodnocování redukčního kritéria

U jednoduchých vrcholů se používá kritérium vzdálenosti od průměrné roviny. Průměrnou rovinou rozumíme rovinu, která je určena všemi sousedními vrcholy aktuálně



Obrázek 2: Úhel určující důležitost hrany, Zdroj: [14]

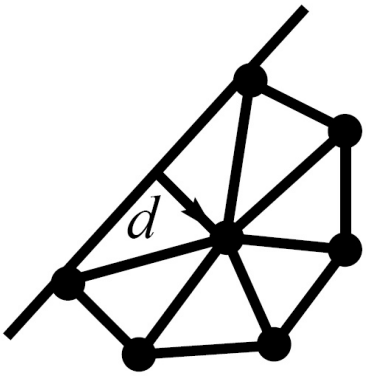


Obrázek 3: Průměrná rovina decimovaného bodu, Zdroj: [2]

decimovaného vrcholu. Ukázka takovéto roviny je na obrázku 3. Pokud je vrchol v určité vzdálenosti od průměrné roviny, může být vymazán. V opačném případě je ponechán.

U hraničních vrcholů a vrcholů na vnitřní hraně se využívá kritéria vzdálenosti od hrany, která je vyobrazena na obrázku 4. V tomto případě algoritmus počítá vzdálenost od úsečky, která je tvořena dvěma body tvořící hranici či důležitou hranu. Pokud je vzdálenost menší než zadaný parametr, vrchol může být vymazán.

Není vždy žádoucí ponechávat důležité hrany. Například mohou sítě obsahovat oblasti relativně malých trojúhelníků, které svírají velké úhly a relativně málo přispívají geometrické aproximaci. Nebo mohou být malé trojúhelníky výsledkem ruchu v originální síti. V těchto případech rohové vrcholy, které se obvykle neodstraňují a vrcholy na



Obrázek 4: Vzdálenost od hranice sítě, Zdroj: [2]

vnitřní hraně, které se ohodnocují pomocí vzdálenosti od hrany, mohou být ohodnoceny pomocí vzdálenosti od roviny. Nazýváme to zachovávání hran a jedná se o uživatelsky nastavitelný parametr.

Pokud může být vrchol smazán, cyklus vrcholů vytvořený odstraněním okolních trojúhelníků se musí triangulovat. Pro vrcholy na vnitřní hraně se musí cyklus rozdělit na dvě poloviny, pomocí dělící úsečky, která spojuje vrcholy tvořící důležitou hranu. Pokud se cyklus takto rozdělit dá, například když se dva výsledné cykly nepřekrývají, je rozdělen a každý nově vzniklý cyklus je triangulován zvlášť.

2.2.4 Triangulace

Odstranění vrcholu vytváří jeden (pro jednoduchý a hraniční vrchol) nebo dva cykly (pro vrchol na vnitřní hraně). V každém cyklu musí být vytvořena triangulace, u které nedochází k překrývání a degeneraci trojúhelníků. Navíc je vhodné vytvořit trojúhelníky s dobrým vzhledovým koeficientem a takové, které se podobají původnímu cyklu, jak je to jen možné.

V zásadě lze říci, že nelze použít dvou rozměrný algoritmus pro vytvoření triangulace, jelikož cyklus většinou není planární. Navíc jsou zde dvě důležité vlastnosti cyklu, které

mohou být použity výhodně. Za prvé, pokud se cyklus nedá triangulovat, vrchol, který tento cyklus vytváří, nesmí být odstraněn. Za druhé, jelikož jsou všechny smyčky ve tvaru hvězdy, triangulační postupy založené na rekurzivním dělení cyklu jsou efektivní. Jakmile je triangulace ukončena, původní vrchol a cyklus trojúhelníků je odstraněn. Podle Eulerova vztahu vyplývá, že odstraněním jednoduchého vrcholu, rohového vrcholu nebo vrcholu na vnitřní hraně se počet trojúhelníků v síti sníží právě o dva. Pokud je odstraněn hraniční vrchol, počet trojúhelníků se sníží právě o jeden.

2.3 Decimace hran [3]

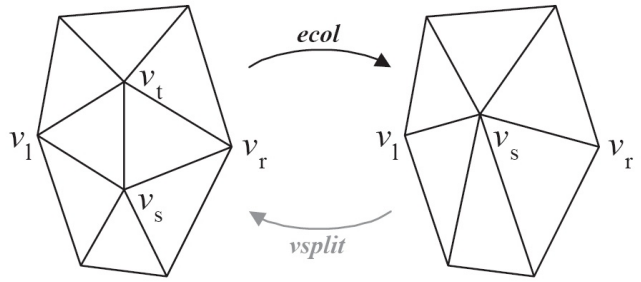
V roce 1993 Hugues Hoppe a další spolupracovníci prezentovali metodu optimalizace sítě založené na opakovaném používání tří základních transformací: zhroucení hrany, rozdělení hrany a prohození hrany [4]. O několik let později v roce 1996 byl stejným autorem prezentován další článek, který na tuto myšlenku bezprostředně navazoval [3]. Bylo totiž zjištěno, že pro efektivní simplifikační metodu nad danou trojúhelníkovou sítí je zapotřebí pouze jedné z těchto základních transformací a to „zhroucení hrany“. Za tímto účelem navíc definoval novou datovou strukturu pro popsání trojúhelníkové sítě s několika úrovněmi detailu, kterou nazval progresivní reprezentace sítě.

2.3.1 Progresivní reprezentace sítě

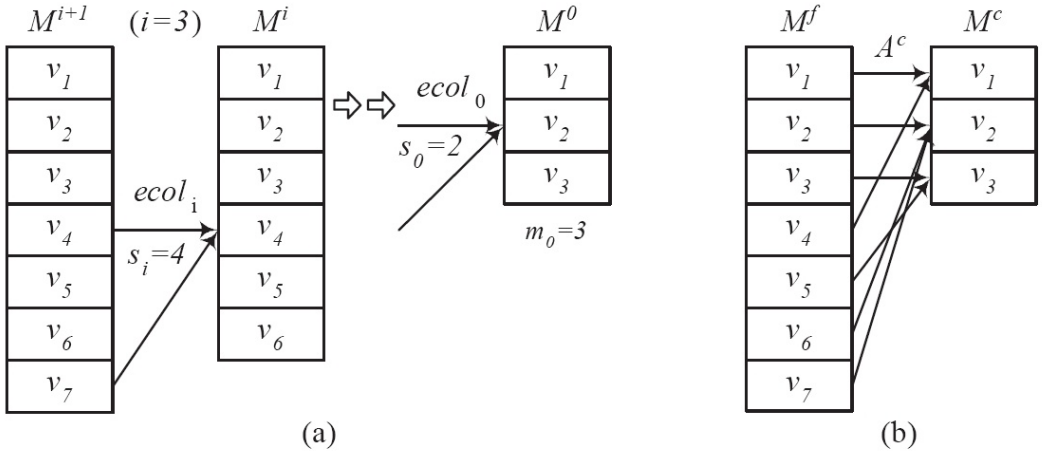
Jak je ukázáno na obrázku 5, transformace zhroucení hrany $ecol(\{v_s, v_t\})$ sjednocuje dva incidentní vrcholy v_s a v_t do jediného vrcholu v_s. Vrchol v_t a dva incidentní trojúhelníky $\{v_s, v_t, v_l\}$ a $\{v_s, v_t, v_r\}$ během procesu zmizí. Pro nový sjednocený vrchol v_s je specifikována poloha.

Takto může být počáteční síť $\widehat{M} = M^n$ zjednodušena na hrubší síť M^0 za použití posloupnosti n úspěšných zhroucení hrany:

$$(\widehat{M} = M^n) \overset{ecol_{n-1}}{\rightarrow} \dots \overset{ecol_1}{\rightarrow} M^1 \overset{ecol_0}{\rightarrow} M^0 \tag{2}$$



Obrázek 5: Transformace zhroucení hrany, Zdroj: [3]



Obrázek 6: (a) Posloupnost hroucení hran; (b) Výsledná sousednost vrcholů, Zdroj: [3]

Konkrétní posloupnost transformací zhroucení hrany musí být vybírána velice obezřetně, jelikož určuje kvalitu aproximované sítě $M^i,\ i < n$.

Nechť je m_0 počet vrcholů v M^0, nechť jsou vrcholy sítě M^i označeny jako $V^i = \{v_1, \ldots, v_{m_0+i}\}$ tak, že hrana $\{v_{s_i}, v_{m_0+i+1}\}$ se hroutí při $ecol_i$ jak je tomu na obrázku 6. Jelikož mohou mít vrcholy v různých sítích různou polohu, označíme polohu vrcholu v_j v M^i jako v_j^i.

Klíčovým pozorováním však je, že transformační operace zhroucení hrany má svou proti operaci. Nazvěme tedy opačnou operaci rozdělení vrcholu, jak je naznačeno na

obrázku 5. Transformace rozdělení vrcholu $vsplit(s,\ l,\ r,\ t,\ A)$ přidá v blízkosti vrcholu v_s nový vrchol v_t a dva nové trojúhelníky $\{v_s, v_t, v_l\}$ a $\{v_s, v_t, v_r\}$. Pokud je hrana $\{v_s, v_t\}$ hranicí sítě, ponecháme $v_r = 0$ a je přidán pouze jeden trojúhelník. Transformace také aktualizuje atributy sítě v okolí, kde je transformace prováděna. Tyto informace dané A obsahují polohy vrcholů v_s a v_t, diskrétní atributy $d_{\{v_s,v_t,v_l\}}$ a $d_{\{v_s,v_t,v_r\}}$ dvou nových trojúhelníků a skalární atributy ovlivněných rohů.

Jelikož jsou operace zhroucení hrany vratné, můžeme reprezentovat obyčejnou trojúhelníkovou síť $\widehat{M}$ jako jednoduchou síť M^0 společně s posloupností n $vsplit$ záznamů:

$$M^0 \overset{vsplit_0}{\rightarrow} M^1 \overset{vsplit_1}{\rightarrow} \dots \overset{vsplit_{n-1}}{\rightarrow} (M^n = \widehat{M}) \tag{3}$$

Kde každý takovýto záznam je parametrizován jako $vsplit(s_i,\ l_i,\ r_i,\ A_i)$. Nazýváme tedy $(M^0,\ \{vsplit_0, \dots, vsplit_{n-1}\})$ progresivní reprezentací (PM) sítě $\widehat{M}$.

2.4 Porovnání popsaných metod

V následující kapitole budou shrnuty vlastnosti redukčních metod popsaných v předešlých kapitolách 2.1,2.2 a 2.3 a následně vybrána jedna metoda, kterou budeme implementovat.

2.4.1 Shlukování vrcholů

Největší výhodou této metody je možnost redukce předem naprosto nepřipravené sítě. Síť je vždy na vstupu pouze rozdělena na několik clusterů, a potom iterativně dělena, dokud není dosaženo požadované redukce (viz. kapitola 2.1.1). Jednou z nevýhod této metody je však vyšší paměťová a časová náročnost při výpočtech během samotného redukčního a triangulačního procesu.

2.4.2 Decimace hran

Samotná progresivní reprezentace sítě je velice paměťově náročný prvek pro uchování během redukčního procesu. Každá jednotlivá transformace zhroucení hrany je popsána několika dalšími datovými strukturami (viz. 2.3.1). I když výsledky aproximovaných sítí bývají kvalitnější, pro naše účely z hlediska paměťové náročnosti je tato metoda nevyhovující.

2.4.3 Decimace vrcholů

U Schroederovy metody, na rozdíl od předchozího shlukování vrcholů, je nutné před zpracování sítě. Každý bod sítě musí být zařazen do jedné z topologií, čímž mu je přiděleno redukční kritérium, podle kterého se bude hodnotit jeho důležitost v síti (viz. kap. 2.2.2 a 2.2.3). Samotný redukční algoritmus však už nevyžaduje náročné výpočty, pouze porovnává prioritu určitého bodu s hraničním redukčním kritériem (viz. kap. 2.2.3). Poté, co je vrchol odstraněn, je vzniklý cyklus okolních bodů lokálně triangulován (viz. kap. 2.2.4). Metoda tedy není příliš paměťově náročná a po určení topologie bodů, by neměla být náročná ani časově.

2.4.4 Zvolená metoda

Zvolit efektivní a rychlý redukční algoritmus je obtížné a závislé na celkové topologii, tvaru a rozsáhlosti trojúhelníkové sítě. Pro naše účely jsme vybrali redukční algoritmus založený na decimaci vrcholů (tzv. Schroederova metoda viz. kapitola 2.2) s určením důležitosti vrcholů pomocí metrické metody délky průměrné normály, která bude blíže popsána v kapitole 4.3.1. Daný algoritmus pro ohodnocování důležitosti vrcholu se podle literatury jeví jako velice efektivní a samotný druh redukce za pomocí decimace vrcholů jako rychlý a účinný algoritmus.

Nevýhodou této skupiny algoritmů jsou méně kvalitní aproximace oproti metodám založených na odlišných principech, jako je například shlukování vrcholů, či decimace

hrany, které jsou však mnohem více paměťově náročné. Daný algoritmus i model datových struktur prstencového typu, který je popsán v kapitole 3.2, jsme volili s přihlédnutím na použití a účel aplikace.

Redukční algoritmus založený na decimaci vrcholů je jedním z nejstarších algoritmů sloužících pro zjednodušování polygonálních sítí a je tedy velmi známý a relativně jednodušší než jiné algoritmy.

3 Návrh implementace

V této části práce se budeme zabývat návrhem implementace aplikace, která bude ilustrovat práci vybrané redukční metody. Návrh se skládá z popisu použitých technologií a návrhu datových struktur, které budou použity pro účel aplikace.

3.1 Použité technologie

Pro realizaci datových struktur jsme se rozhodli použít programovací jazyk C# na platformě .NET Framework. Jedním z hlavních důvodů tohoto rozhodnutí byl snadný management paměti oproti nativnímu kódu psaném například v C++. Jsme si vědomi, že touto volbou potlačujeme multiplatformnost celé aplikace, ale věříme, že pozitivní vlastnosti, které s sebou přináší jazyk C# a celá platforma, tuto nevýhodu vyváží.

Samotné uživatelské rozhraní bude vyvíjeno pomocí knihovny Qt a $QGLViewer$. Qt je velice známá multiplatformní knihovna pro vývoj aplikací s grafickým uživatelským rozhraním. Knihovna podporuje nejenom vytváření oknových aplikací, ale také lokalizaci aplikací, SQL, zpracování XML, správu vláken a přístup k souborům. Knihovna $QGLViewer$ je volně šiřitelná knihovna, která spojuje knihovnu Qt s $openGL$. Základní třída celé knihovny souhlasně pojmenovaná $QGLViewer$ je odvozena z jedné ze základních tříd knihovny Qt $QWidget$, která představuje hlavní okno aplikace. Tato vlastnost velice usnadňuje propojení obou knihoven a snadné vkládání jednotlivých oken do sebe.

Pro propojení datových struktur a uživatelského rozhraní použijeme CLR (Common Language Runtime). CLR je jádrová komponenta vytvořená iniciativou Microsoft .NET. Tato komponenta by se dala označit jako vlastní implementace CLI (Common Language Infrastructure) standardu firmou Microsoft, který definuje prostředí pro vykonávání kódu programu. CLR je spouštěn ve formě byte kódu, který se nazývá Common Intermediate Language (CIL), dříve známý jako MSIL (Microsoft Intermediate Language).

3.2 Datové struktury

3.2.1 Datový model

Pro naši aplikaci jsme se rozhodli použít takzvaný prstencový datový model polygonální sítě [2]. Základními stavebními kameny jsou datové struktury vrchol (vertex) a ploška, neboli trojúhelník (face). Hrana nemusí být definována konkrétně, pouze jako pár vrcholů, u kterého nezáleží na pořadí. Samotná trojúhelníková síť pak bude popsána pomocí seznamu vrcholů a trojúhelníků, které nám takovouto síť jednoznačně určují. Daný návrh, ze kterého vycházíme, se jeví jako velice efektivní a jednoduchý, obzvláště pro práci s redukčním algoritmem, který jsme zvolili pro implementaci. O zvoleném redukčním algoritmu budeme pojednávat v kapitole 4. Známou nevýhodou zvolené datové struktury je vyšší paměťová náročnost než u jiných datových struktur, jako je například *winged edge*, u které lze přesně určit velikost, kterou síť zabere v paměti v závislosti na počtu vrcholů a trojúhelníků. Věříme, že pozitivní vlastnosti prstencové datové struktury převáží její paměťovou náročnost.

3.2.2 Vrchol

Datová struktura vrchol musí obsahovat jednoznačné určení polohy v třírozměrném prostoru (souřadnice). Vrchol bude dále také obsahovat seznam trojúhelníků, ve kterých se daný vrchol nachází a také seznam hran incidentních s tímto vrcholem. Seznam hran není naprosto nezbytný, ale urychluje práci při určení lokální topologie vrcholu a hledání sousedů daného vrcholu. Jelikož seznam hran není potřebný pro průběh samotného redukčního algoritmu, může být paměť, která je potřebná pro uchování tohoto seznamu, po zjištění topologie uvolněna. Jedním z hlavních důvodů stálého udržování seznamu incidentních hran je rychlejší průchod při hledání cyklu vrcholů kolem redukovaného vrcholu. Nalezení cyklu však může být provedeno i za pomoci seznamu trojúhelníků incidentních k danému vrcholu. U vrcholu budeme evidovat další členské proměnné

spojené s kritériem ohodnocování priority daného vrcholu a další pomocné proměnné pro určení, zda je daný vrchol součástí aktuální sítě. Tedy zda byl redukován nebo zůstává nezměněn. Přehled členských proměnných je vyobrazen ve výpisu zdrojového kódu 1.

```
public enum TOPOLOGY { COMPLEX, BORDER, CORNER, EDGE, SIMPLE, NONE }

public class Vertex
{
    #region Members

    private float [] xyz;
    private List<Edge> edges;
    private List<Face> faces;
    private TOPOLOGY topol;
    private float avgNorm;
    private bool deleted;
    private float prior;

    #endregion
}
```

Výpis 1: Datová struktura vrchol

Metod, které je bezpodmínečně nutné implementovat nad touto strukturou, je hned několik. Metody pro určení lokální topologie bodu zahrnující vytvoření potřebných hran dle incidentních trojúhelníků. Metody pro zjištění priority daného bodu v závislosti na kritériu daného redukčního algoritmu. Metody pro přidání trojúhelníku resp. hrany do seznamu trojúhelníků resp. hran, a to jak přímé, tak s kontrolou možného duplicitního vložení. Dále metody pro realizaci samotného redukčního algoritmu a metody související s triangulací díry vzniklé po odstranění daného bodu, o kterých budeme pojednávat v kapitolách 4.4 a 4.5.

3.2.3 Trojúhelník

Dalším prvkem trojúhelníkové sítě je samotný trojúhelník neboli ploška. Jak již bylo řečeno dříve, použijeme návrh standardního prstencového datového modelu [2], z něhož vyplývá, že trojúhelníky jsou popsány jako trojice indexů ze seznamu vrcholů.

V našem případě budou vrcholy reprezentovány přímo ukazateli (odkazy) na datovou strukturu vrchol, které se v tomto seznamu nacházejí. Kromě odkazů na vrcholy, ze kterých se daný trojúhelník skládá, je nutno evidovat jeho normálu a další členské proměnné. Odkazy na vrcholy a normála jsou nejdůležitějšími členskými proměnnými pro výslednou vizualizaci výsledku v grafické podobě. Trojúhelník musí implementovat metody pro zjištění a výpočet normály. Následující výpis 2 ukazuje možnou implementaci datové struktury trojúhelník.

```
public class Face
{
    #region Members

    private Vertex[] v;
    private Vertex normal;
    private bool deleted;
    private bool justDeleted;
    private bool added;
    private bool justAdded;
    private bool inLoop;
    private bool oriented;

    #endregion
}
```

Výpis 2: Datová struktura trojúhelník

3.2.4 Hrana

Jak již bylo uvedeno výše v kapitole 3.2.1, datová struktura hrany nemusí být podle prstencového datového modelu explicitně definována. Pro naše účely si však hranu popíšeme detailněji, jelikož je stejně důležitá jako obě výše popsané struktury.

Hrana je určena svými koncovými vrcholy, u nichž nezáleží na pořadí. Další členskou proměnnou je seznam trojúhelníků, které jsou incidentní s danou hranou. Jelikož každý vrchol obsahuje seznam všech trojúhelníků, ve kterých leží, jedná se o redundantní data. Avšak pro účely určení lokální topologie daného bodu jsou tyto informace o hraně naprosto nezbytné. U hrany dále evidujeme její důležitost, o jejímž určování budeme

pojednávat v kapitole 4.2. Stejně jako v předchozích případech je ve výpisu 3 zobrazena možná implementace této datové struktury (tedy hrany).

```
public class Edge
{
    #region Members

    private Vertex v1;
    private Vertex v2;
    private List<Face> f;
    private float prior;
    private bool featEdge;
    private bool deleted;
    private bool justDeleted;
    private bool added;
    private bool justAdded;

    #endregion
}
```

Výpis 3: Datová struktura hrana

Hrana musí implementovat metody pro určení její důležitosti v závislosti na daných vstupních parametrech, dále také metody pro přidání trojúhelníku do seznamu incidentních plošek a to opět jak přímé tak s kontrolou již vložených trojúhelníků.

3.2.5 Síť

Celek složený z výše popsaných datových struktur se bude nazývat datová struktura sítě. Jak již bylo řečeno, základem samotné sítě je seznam vrcholů a trojúhelníků, ze kterých se daná síť skládá. Seznam hran je v tomto případě zbytečné uchovávat jako jeden celek, jelikož jeho použití takovýmto způsobem nemá velký význam. Je naprosto postačující, když veškeré odkazy na hrany budou uchovány pouze v jednotlivých seznamech vrcholů, které jsou s nimi incidentní.

U trojúhelníkové sítě musíme dále evidovat mnoho dalších členských proměnných jako je například celkový počet vrcholů a trojúhelníků v síti, počet aktuálně zobrazovaných vrcholů a trojúhelníků, počet redukovaných vrcholů a trojúhelníků atd. Výčet těchto proměnných je uveden ve výpisu 4. Nad touto datovou strukturou musí být implemen-

továny metody umožňující načíst danou síť ze souboru, zjistit lokální topologie všech bodů sítě a redukovat danou síť pomocí námi zvoleného redukčního algoritmu na určitý stupeň aproximace.

```
public class Mesh
{
    #region Members

    private List<Vertex> v;
    private List<Face> f;

    //pocty vrcholu a troj nactenych ze souboru
    private int vFileCount;
    private int fFileCount;

    // pocty aktualne zobrazovanych vrcholu a troj
    private int actVCount;
    private int actFCount;
    private int delVCount;
    private int redVCount;

    //minimalni a maximalni avg Normala
    private float minAvg;
    private float maxAvg;

    //pocty vrcholu dle topologii
    private int simpleV;
    private int edgeV;
    private int cornerV;
    private int borderV;
    private int complexV;

    //cos uhlu pro urceni dulezite hrany
    private float cosAngle;

    //cesta k souboru pro nacteni
    private string path;
    private bool loaded;

    //body pro urceni zobrazovaciho kvadru a stredu meshe
    private Vertex minV;
    private Vertex maxV;
    private Vertex middle;

    #endregion
}
```

Výpis 4: Datová struktura síť

4 Implementace

V této části práce se budeme zabývat implementací aplikace, která bude provádět jeden z redukčních algoritmů a vizualizovat výsledky aproximací pomocí $openGL$. V jednotlivých podkapitolách bude popsána použitá metrická funkce a použitá redukční metoda včetně popisu zvolené triangulační metody.

4.1 Proces redukce

Jak už bylo uvedeno v kapitole 2.2.1, daný redukční algoritmus lze rozdělit do tří základních kroků:

1. charakterizace lokální geometrie a topologie vrcholu
2. ohodnocení dle redukčního kriteria
3. triangulace vzniklé díry

Pro účely implementace si celý redukční proces rozdělíme do čtyř kroků, které samy o sobě mohou představovat jednotlivé samostatné metody:

1. určení lokální topologie bodů
2. ohodnocení dle stanoveného kriteria
3. redukce daného bodu
4. triangulace díry vzniklé po odstranění bodu

V následujících kapitolách si jednotlivé kroky popíšeme podrobněji.

4.2 Určení lokální topologie

Implementačně musíme nejdříve podle seznamu trojúhelníků určit a vytvořit všechny hrany, které jsou s daným bodem incidentní a ke každé hraně přiřadit správné trojúhelníky, které k ní přísluší. Po ukončení této procedury jsme schopni zařadit každý vrchol

do jedné z pěti skupin (dle Schroedera): jednoduchý, na vnitřní hraně, rohový, hraniční, složitý.

Pro určení lokální topologie slouží pravidla pro přiřazení bodu do jedné z pěti skupin vrcholů, jejichž postup byl již podrobně popsán v kapitole 2.2.2, a proto zde nebude opakován. Musíme zde však popsat postup určení důležitosti hrany a z něj vyplývající rozlišení jednoduchého a rohového vrcholu a vrcholu na vnitřní hraně. Ve výpisu 5 je vyobrazena metoda, která podle členského seznamu trojúhelníků vytvoří incidentní hrany k danému vrcholu. Metoda je implementována nad datovou strukturou vrchol.

```
public void FindTopology(bool newEdges)
{
    int fSize = GetFaceCount();
    int vIndex = -10;

    for (int i = 0; i < fSize; i++)
    {
        Face face = GetFace(i);
        vIndex = -10;
        if (!face.Deleted)
        {
            for (int j = 0; j < 3; j++)
            if (this.Equals(face[j]) )
            {
                vIndex = j;
                break;
            }

            for (int j = 1; j < 3; j++)
            {
                Vertex vNext = face[(vIndex + j) % 3];
                Edge e = new Edge(this, vNext);
                e.Added = newEdges;
                e.AddFaceDir(face);
                Edge tmp = AddEdge(e);

                if (tmp.Equals(e) == true)
                    vNext.AddEdgeDir(e);
                else
                    tmp.AddFace(face);
            }
        }
    }
}
```

Výpis 5: Metoda pro vytvoření incidentních hran

Jednoduché vrcholy, vrcholy na vnitřní hraně a rohové vrcholy jsou obklopeny úplným cyklem trojúhelníků a mají tudíž stejný počet hran a trojúhelníků s nimi incidentních. Liší se však v důležitosti jednotlivých hran. Důležitost hrany se z pravidla posuzuje podle velikosti prostorového úhlu mezi trojúhelníky, které jsou s danou hranou incidentní. Tento úhel je určen jako úhel svíraný mezi normálami obou rovin (viz. obr. 2). Pokud tato hodnota překročí určitou hraniční hodnotu, označíme hranu za důležitou a uložíme její prioritu. Pokud z vrcholu nevychází žádná důležitá hrana, jedná se o vrchol jednoduchý. Pokud z vrcholu vychází jedna či tři a více důležitých hran, jedná se o vrchol rohový. Pokud z vrcholu vychází právě dvě důležité hrany, jedná se o vrchol na vnitřní hraně.

Přiřazení správné topologie danému bodu je základním úkolem redukčního algoritmu, bez kterého je samotný decimační proces nesmyslný.

Pokud je u šech bodů určena lokální topologie a geometrie, můžeme ohodnotit body dle daného redukčního kritéria. Pro tyto účely využijeme metrickou funkci velikosti průměrné normály. Jelikož tato metoda zatím v této práci nebyla popsána, následující kapitola je jí věnována. Obsahuje seznámení se základními principy této metody.

4.3 Ohodnocení dle stanoveného kriteria

4.3.1 Metoda průměrné normály

Jak již víme, u klasické Schroederovy metody se složité vrcholy neodstraňují. Hraniční vrcholy jsou ohodnoceny podle kritéria vzdálenosti od hranice sítě. Vrcholy na vnitřní hraně jsou ohodnoceny podle vzdálenosti od úsečky spojující vrcholy tvořící důležité hrany a nakonec vrcholy jednoduché a rohové jsou ohodnocovány dle kritéria vzdálenosti od průměrné roviny, která je určena všemi sousedy odstraňovaného vrcholu.

Metrická metoda velikosti průměrné normály zachovává všechny původní způsoby ohodnocování důležitosti vrcholů, až na skupinu jednoduchých vrcholů. Jednoduché vrcholy jsou ohodnocovány podle již zmiňované délky průměrné normály. Jedná se v podstatě o vypočítání průměrné normály z normalizovaných normálových vektorů

trojúhelníků incidentních s daným vrcholem a následný výpočet velikosti tohoto vektoru. Ve výpisu 6 je znázorněna metoda určující velikost takovéto průměrné normály daného bodu. Velikost průměrné normály je v intervalu $< 0, 1 >$, kde 0 je hodnota nejnižší důležitosti.

```
public float CountAvgNormal()
{
    int size = GetFaceCount();
    Vertex tmp = new Vertex();
    for ( int i=0; i<size; i++ )
    if ( !faces[i ]. Deleted)
        tmp += faces[i ]. Normal;

    tmp /= (float) size;
    avgNorm = tmp.GetMagnitude();
    return avgNorm;
}
```

Výpis 6: Metoda pro určení velikosti průměrné normály

4.3.2 Použitá metrická funkce

Pro účely naší aplikace jsme ještě poněkud pozměnili danou metrickou metodu, a to následujícím způsobem.

Jednoduché a rohové vrcholy budeme ohodnocovat pomocí délky průměrné normály. Vrcholy na vnitřní hraně podle zmiňované vzdálenosti od úsečky s koncovými vrcholy, které tvoří důležité hrany vycházející z redukovaného vrcholu.

Jelikož se u naší aplikace předpokládá práce s uzavřenými modely, které obsahují pouze $manifold$ povrchy, neměly by takovéto sítě obsahovat žádné hraniční ani složité vrcholy. Pokud takovéto vrcholy obsahují, jejich počet je velmi malý. Z tohoto důvodu jsme se rozhodli tyto skupiny vrcholů nedecimovat a přiřadit jim maximální prioritu. Pokud se takovéto vrcholy v síti objeví, nebudeme je redukovat a ponecháme je v síti.

Jednoduché vrcholy mají prioritu nejnižší, dále potom vrcholy na vnitřní hraně a rohové vrcholy. Jelikož bývá jednoduchých vrcholů v síti nejvíce, naše rozhodnutí nedecimovat hraniční vrcholy nijak nesnižuje účinnost redukčního algoritmu. Pokud jsou

všechny vrcholy ohodnoceny dle příslušného kritéria a je každému bodu přiřazena priorita, můžeme započít se samotnou decimací vrcholů.

4.4 Redukce daného bodu

Před samotnou decimací musíme vždy vrcholy setřídit dle priority. Při každém kroku je vybrán vrchol, který má být decimován a po jeho odstranění se musí triangulovat vzniklá díra. Redukční cyklus je ukončen po dosažení určitého kritéria, kterým může být například určitá procentuální aproximace, či počet zobrazovaných trojúhelníků.

Každý decimační krok musí začít vytvořením uzavřené smyčky kolem decimovaného vrcholu. Proto projdeme všechny hrany, které jsou incidentní s daným vrcholem a pokud není daná hrana smazána, přidáme sousední vrchol do cyklu kolem redukovaného vrcholu.

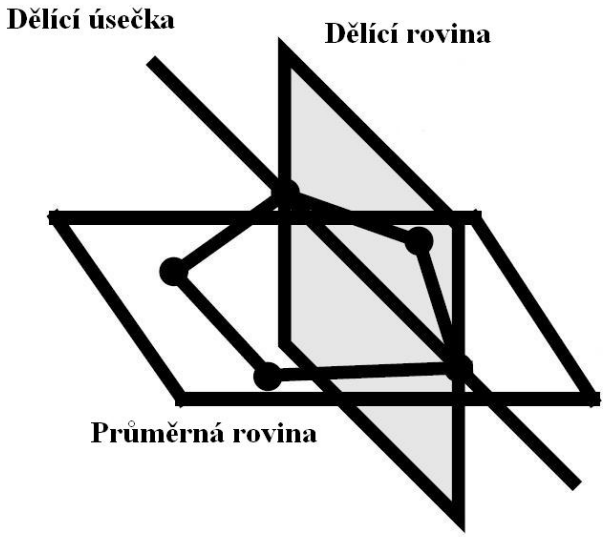
Jakmile je smyčka vytvořena, pokusíme se ji triangulovat. Proces triangulace je popsán v kapitole 4.5. Pokud triangulace proběhla úspěšně, můžeme daný bod označit jako smazaný a rovněž všechny trojúhelníky a hrany s ním incidentní. Poté musíme vytvořit všechny nové hrany podle přidaných trojúhelníků a ověřit topologii všech bodů v cyklu.

Následně aktualizovat hodnoty v síti, jako je počet aktuálně zobrazovaných trojúhelníků, počet redukovaných vrcholů atd. Při každém odstranění jednoduchého vrcholu, vrcholu na vnitřní hraně, či rohového vrcholu se počet trojúhelníků v síti sníží právě o dva.

Pokud byl redukční krok úspěšně dokončen, zkontrolujeme, zda-li jsme dosáhli ukončovací podmínky. Pokud ne, pokračujeme v další iteraci.

4.5 Triangulace vzniklé díry

Triangulaci můžeme provést několika rozličnými způsoby. Jedním z možných způsobů je rekurzivní dělení smyčky určené pro triangulaci podle dělící úsečky [2]. Tato metoda triangulace je relativně jednoduchá a překvapivě efektivní, i když se jedná o re-



Obrázek 7: Znázornění dělící a průměrné roviny, Zdroj: [2]

kurzivní metodu. Je to dáno obvyklým tvarem triangulovaného cyklu, který má přibližný tvar hvězdy.

Na začátku každého triangulačního procesu se musí určit dělící úsečka. Úsečka může být tvořena libovolnými dvěma nesousedními vrcholy. U vrcholů na vnitřní hraně se preferuje dělící úsečka tvořená vrcholy, které dříve určovaly důležité hrany. Pokud jsme určili dělící úsečku, můžeme každou polovinu vzniklou po rozdělení touto úsečkou triangulovat zvlášť. Takto provádíme dělení na menší smyčky, dokud smyčka neobsahuje pouze tři vrcholy, které jsme schopni triangulovat pomocí jednoho trojúhelníku.

Pro dělení smyčky opět existuje více metod. Můžeme například použít metodu dělící roviny. Metoda dělící roviny je založena na vytvoření roviny kolmé k průměrné rovině, která navíc prochází dělící úsečkou. Takováto rovina je znázorněna na obrázku 7. Průměrná rovina je určena všemi body smyčky. Musíme však brát na vědomí, že jelikož je smyčka velice často neplanární, body smyčky v dané průměrné rovině většinou neleží.

Rovina kolmá k takovéto průměrné rovině nám rozdělí prostor na dva poloprostory. Pro každý vrchol ve smyčce určíme, do kterého z těchto dvou poloprostorů patří a zařadíme ho do správné skupiny bodů pro následující triangulaci. Každá takováto triangulace může být kvalitativně ohodnocena dle určitých kritérií.

Je zřejmé, že se budeme snažit provést co nejlepší triangulaci, která je co nejvíce vizuálně podobná původnímu cyklu trojúhelníků před decimací daného vrcholu. Ohodnocení triangulace může být provedeno například za pomocí určení vzdáleností všech vrcholů od dělící roviny děleno délkou dělící úsečky [2]. Čím menší jsou rozdíly vzdáleností jednotlivých vrcholů od roviny, tím je triangulace kvalitnější.

Jedním z dalších způsobů ohodnocení kvality vytvořené triangulace je za pomocí určení úhlů u příslušných vrcholů všech trojúhelníků. Čím menší jsou rozdíly vnitřních úhlů v trojúhelníku, tím je triangulace kvalitnější.

5 Porovnání výsledků a jejich vizualizace

Stěžejním předmětem této kapitoly bude ohodnocení námi implementovaného redukčního algoritmu. Algoritmus budeme hodnotit jak z hlediska jeho časové náročnosti, tak z hlediska kvality produkovaných aproximací.

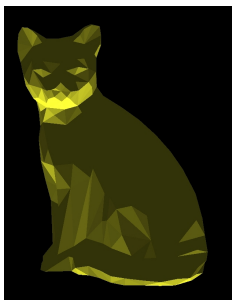
Nejprve budeme seznámeni s datovými modely, které byly použity pro testování našeho algoritmu. Poté se pokusíme časovou náročnost algoritmu porovnat s jinými známými redukčními metodami a na závěr kapitoly porovnáme aproximace testovaných modelů s aproximacemi, které byly výstupem naší aplikace.

5.1 Použité modely

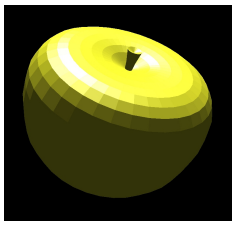
Modely, které byly použity pro testování naší aplikace, jsou veřejně dostupné na mnoha internetových stránkách institutů a organizací zabývajících se počítačovou grafikou či informatikou obecně. Všechny námi použité trojúhelníkové modely jsme získali na stránkách University of Princeton [13] ve standardním ply formátu a jsou vyobrazeny na obrázcích 8-15.

V tabulce 1 se pokusíme stručně popsat použité trojúhelníkové sítě. Je zde uveden orientační název sítě a také počet vrcholů a trojúhelníků v počátečním neredukovaném stavu. Dále jsou zde uvedeny počty vrcholů dle příslušných topologií (viz. 2.2.2) tak, jak byly rozděleny naší aplikací. Jednoduchý vrchol (Simple), vrchol na vnitřní hraně (Edge), vrchol rohový (Corner), vrchol hraniční (Border) a vrchol komplexní (Complex).

Jak je vidět z tabulky 1, použili jsme modely s velmi rozličnou velikostí (počtem plošek). Od malých modelů obsahujících pouze několik tisíc trojúhelníků až po modely relativně velké, které se skládají až z jednoho milionu trojúhelníků. Všechny modely uvedené v tabulce 1 byly redukovány pomocí naší aplikace.



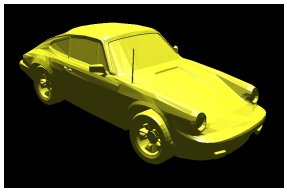
Obrázek 8: Cat (671 plošek), Zdroj: vlastní aplikace



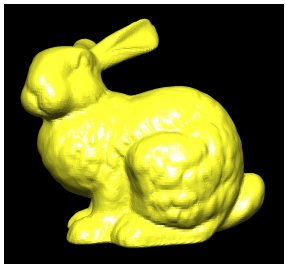
Obrázek 9: Apple (1 704 plošek), Zdroj: vlastní aplikace

Obrázek 10: Cow (5 804 plošek), Zdroj: vlastní aplikace

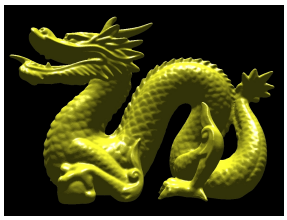
Obrázek 11: Hind (6 448 plošek), Zdroj: vlastní aplikace



Obrázek 12: Porsche (10 474 plošek), Zdroj: vlastní aplikace



Obrázek 13: Bunny (69 451 plošek), Zdroj: vlastní aplikace



Obrázek 14: Dragon (87 414 plošek), Zdroj: vlastní aplikace

Obrázek 15: Happy (1 087 716 plošek), Zdroj: vlastní aplikace

		Vrcholy					
Název modelu	**Plošky**	**Vše**	**Simple**	**Edge**	**Corner**	**Border**	**Complex**
Cat	671	352	307	9	6	31	0
Apple	1 704	867	819	24	0	24	0
Cow	5 804	2 903	2 793	44	66	0	0
Hind	6 448	3 218	2 243	608	367	0	0
Porsche	10 474	5 247	4 503	310	432	0	2
Bunny	69 451	34 834	34 581	10	20	223	0
Dragon	871 414	437 645	433 348	1	26	4 270	0
Happy	1 087 716	543 652	541 216	1 263	1 173	0	0

Tabulka 1: Seznam použitých modelů a jejich stručný popis

5.2 Ohodnocení algoritmu

Z důvodu vypracování ohodnocení našeho redukčního algoritmu z hlediska časové náročnosti jsme provedli několik měření. Měření byla prováděna na každém modelu samostatně a odečtené časy byly zaznamenány do tabulky 2. Každé měření se skládalo z několika částí:

1. načtení modelu ze souboru (ply) do operační paměti
2. určení lokální topologie všech bodů sítě
3. třídění vrcholů dle priority
4. redukční algoritmus

Měření pro redukční algoritmus byla prováděna vždy pro 80%, 50% a 20% redukci. Jinými slovy byly odečteny časy nutné pro vytvoření aproximací, které obsahují 80%, 50% a 20% z původního počtu trojúhelníků. Jelikož pro modely s malým počtem plošek jsou naměřené časy velice krátké, mohlo dojít k větší chybě a hodnoty uvedené v tabulce 2 jsou pro tyto modely spíše orientační.

V následující tabulce 3 jsou uvedeny podrobnější informace o jednotlivých procentuálních aproximacích. U každého modelu a jeho aproximace je uveden počet vrcholů a plošek, ze kterých se redukovaný model skládá.

				Redukce [s]		
Název modelu	**Načtení [s]**	**Topologie [s]**	**Třídění [s]**	**80%**	**50%**	**20%**
Cat	0.010	0.012	0.002	0.039	0.036	0.055
Apple	0.019	0.024	0.002	0.052	0.069	0.134
Cow	0.040	0.062	0.004	0.163	0.348	0.576
Hind	0.042	0.065	0.005	0.206	0.430	0.748
Porsche	0.075	0.128	0.007	0.275	0.628	1.073
Bunny	0.760	0.803	0.049	1.964	4.876	8.190
Dragon	9.419	9.605	1.180	24.788	63.689	113.067
Happy	13.169	12.691	1.517	29.838	80.715	142.949

Tabulka 2: Naměřené časy pro jednotlivé modely, části algoritmu a stupně redukce

	Redukce 80%		**Redukce 50%**		**Redukce 20%**	
Název modelu	**Plošky**	**Vrcholy**	**Plošky**	**Vrcholy**	**Plošky**	**Vrcholy**
Cat	535	284	335	184	157	95
Apple	1 362	696	852	441	340	185
Cow	4 642	2 322	2 902	1 452	1 160	581
Hind	5 158	2 573	3 224	1 606	1 730	859
Porsche	8 378	4 199	5 236	2 628	2 094	1 057
Bunny	55 559	27 888	34 725	17 471	13 889	7 053
Dragon	697 130	350 503	435 706	219 791	174 282	89 079
Happy	870 172	434 880	543 858	271 723	217 542	108 565

Tabulka 3: Počty zobrazovaných vrcholů a plošek u jednotlivých aproximací

Nyní musíme implementovaný algoritmus porovnat s jinými redukčními algoritmy, abychom byli schopni určit jeho rychlost. Jelikož jsme při implementaci algoritmu provedli určité změny na způsobu ohodnocování (viz. kap. 4.3), nelze rychlost a kvalitu algoritmu přímo srovnávat s ryzí Schroederovou metodou. Pokusíme se však v tabulce 4 srovnat rychlosti podobně založených algoritmů, ale i algoritmů naprosto odlišných.

Jelikož se nám nepodařilo najít srovnání rychlostí pro všechny použité modely, museli jsme se spokojit pouze s několika. V tabulce 4 jsou srovnány rychlostní testy pro Schroederovu metodu[1], R-Simp algoritmus, který byl popsán v kapitole 2.1 a jemu podobný QSlim algoritmus. Oba tyto algoritmy jsou podrobně popsány v [1].

[1]Implementace této metody opět není úplně ryzí.[12]

Všechny časy uvedené v tabulce 4 jsou měřeny pro přibližně 90% redukci daného modelu. Zdůrazňujeme přibližně, jelikož se nám nepodařilo získat aproximace, které byly pro daný model redukovány na přesně stejnou hodnotu.

Název modelu	Schroederova metoda	R-Simp	QSlim	Vlastní metoda
Bunny	13.50	1.65	7.74	8.97
Dragon	93.60	19.72	128.43	140.31
Happy	118.30	24.52	141.12	177.91

Tabulka 4: Porovnání naměřených časů redukce

5.3 Srovnání výsledných aproximací

Potom, co jsme porovnali námi implementovaný algoritmus z hlediska rychlosti jeho vykonávání, musíme ohodnotit také samotné výsledky (výstupy) tohoto algoritmu. Pro určení kvality námi vygenerovaných aproximací je potřeba tyto redukované modely s něčím srovnat. Za tímto účelem jsme se rozhodli použít aproximace vygenerované pomocí aplikace $Mesh\ Simplification\ Viewer$ vytvořené p. Jeffem Sommersem.[2] V aplikaci $Mesh\ Simplification\ Viewer$ je implementován redukční algoritmus zhroucení hrany, který byl popsán v kapitole 2.3, konkrétně algoritmus rozšířený o metrickou funkci kvadratické chyby prezentovaný v [6]. Další metrická funkce použitá v této aplikaci je založena na odstraňování nejkratších hran.

Za pomocí výše zmíněné aplikace jsme schopni vyprodukovat aproximace, kde počet zobrazovaných trojúhelníků skoro přesně souhlasí s počtem trojúhelníků v námi vytvořených redukovaných modelech. Z tohoto důvodu je toto srovnání zásadní, jelikož můžeme porovnat identické aproximace pouze vytvořené pomocí jiných metod.

Následující obrázky se pokusí ilustrovat vzájemnou podobu a rozdíly daných aproximací a tímto způsobem určit, zda-li je náš algoritmus kvalitní či nikoli.

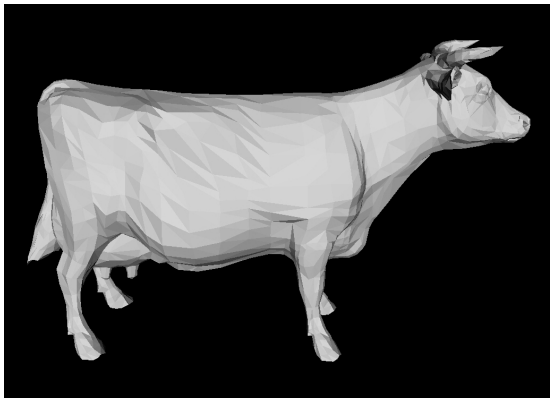
Pro srovnání kvality aproximací jsme si vybrali modely Cow a $Happy$ viz. obrázky 10 a 15. Každý model je vždy srovnán s výše uvedenou metodou zhroucení hrany za použití

[2] Tuto aplikaci je možno získat na adrese: http://jsomers.com/vipm_demo/meshsimp.html[15]

metrické funkce kvadratické chyby a také s touto metodou za použití metrické funkce nejkratší hrany. U každého modelu je porovnáno několik stupňů redukce.

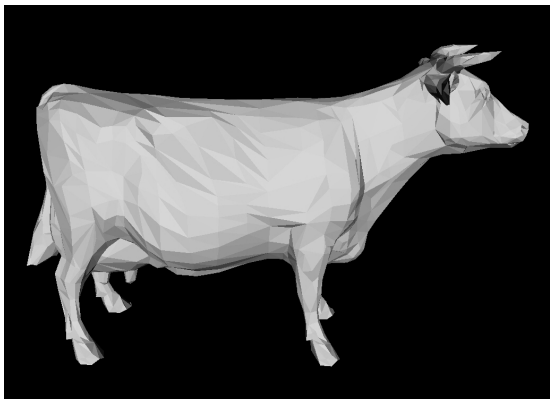
Jelikož jsme vzhledem k rozměrnosti obrázků nemohli zobrazit všechny zpracované modely a jejich aproximace, nachází se tyto v příloze **??**.

Obrázek 16: Cow: 80% redukce (4 642 plošek), Zdroj: vlastní aplikace

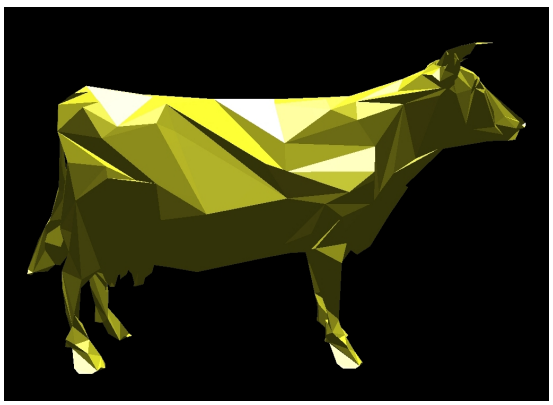


Obrázek 19: Cow: 80% redukce (Quadric error) (4 652 plošek), Zdroj: [15]

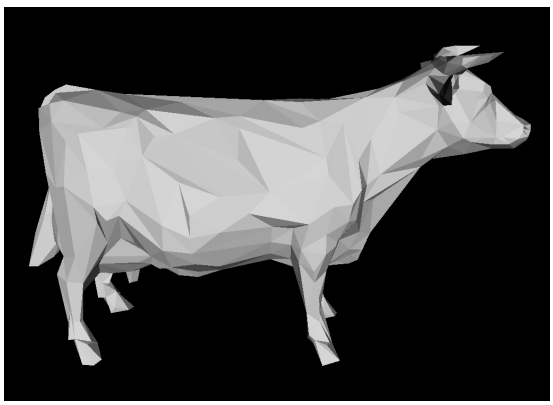
Obrázek 17: Cow: 50% redukce (2 902 plošek), Zdroj: vlastní aplikace



Obrázek 20: Cow: 50% redukce (Quadric error) (2 924 plošek), Zdroj: [15]

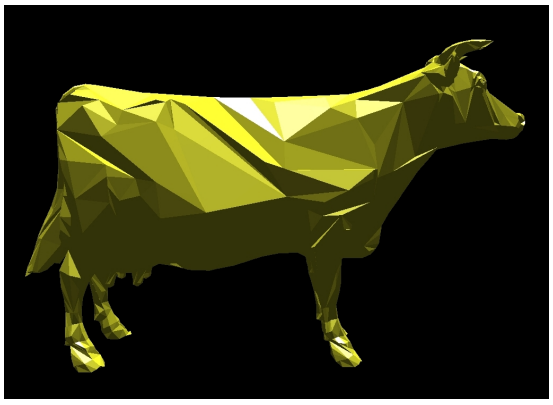


Obrázek 18: Cow: 20% redukce (1 160 plošek), Zdroj: vlastní aplikace

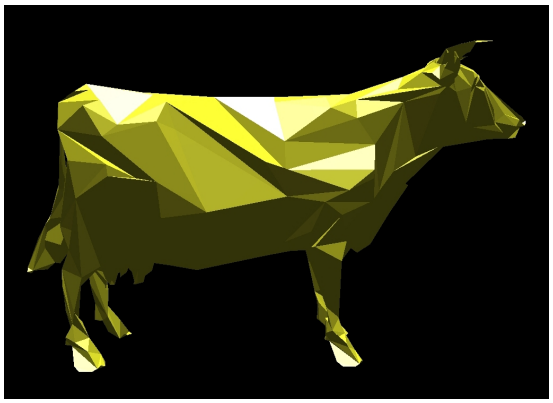


Obrázek 21: Cow: 20% redukce (Quadric error) (1 196 plošek), Zdroj: [15]

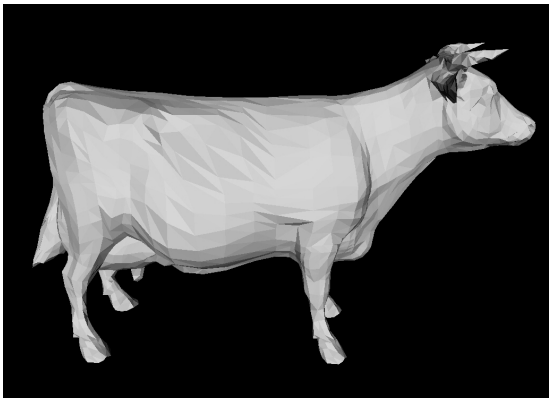
Obrázek 22: Cow: 80% redukce (4 642 plošek), Zdroj: vlastní aplikace



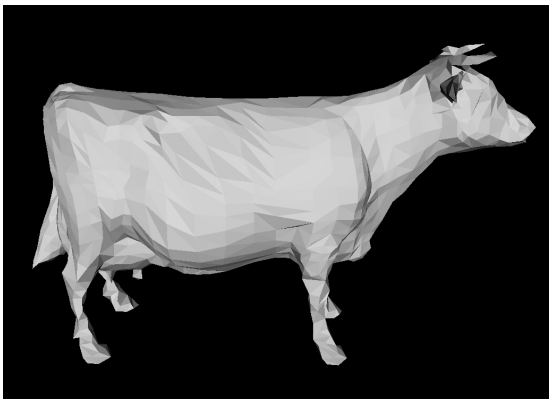
Obrázek 23: Cow: 50% redukce (2 902 plošek), Zdroj: vlastní aplikace



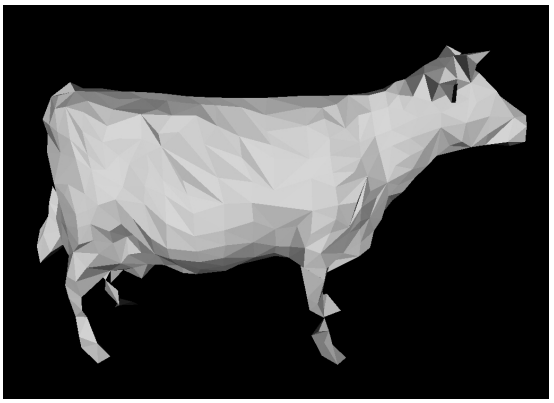
Obrázek 24: Cow: 20% redukce (1 160 plošek), Zdroj: vlastní aplikace



Obrázek 25: Cow: 80% redukce (Shortest edge) (4 660 plošek), Zdroj: [15]

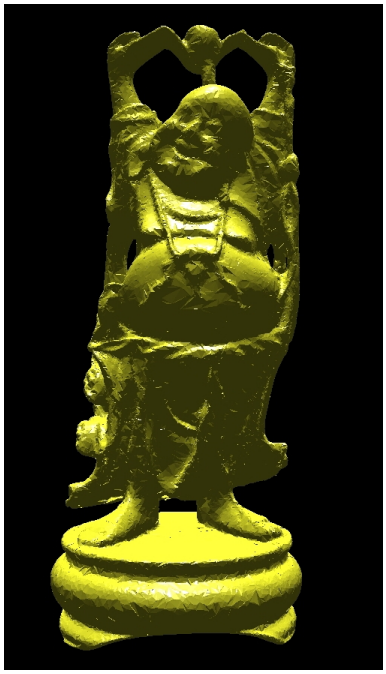


Obrázek 26: Cow: 50% redukce (Shortest edge) (2 942 plošek), Zdroj: [15]

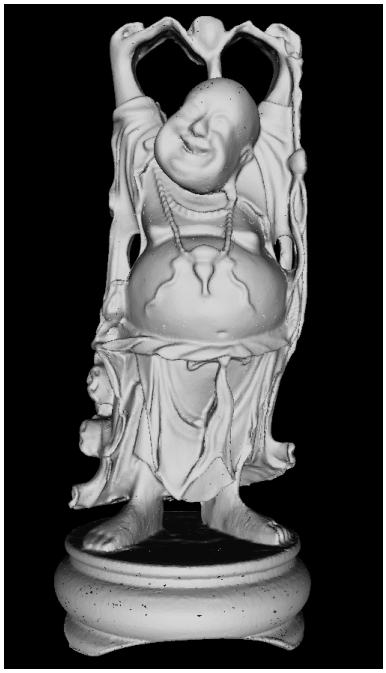


Obrázek 27: Cow: 20% redukce (Shortest edge) (1 204 plošek), Zdroj: [15]

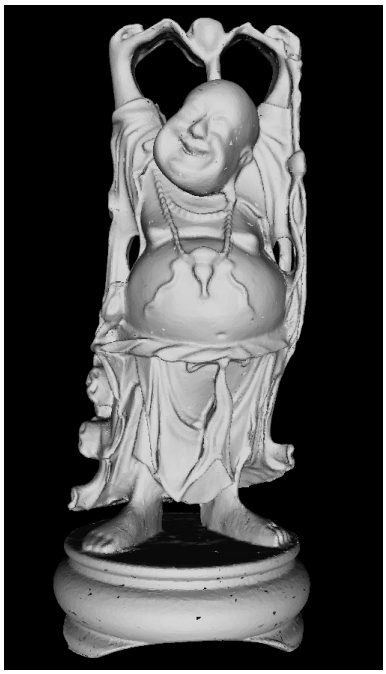
Obrázek 28: Happy: 50% redukce (543 858 plošek), Zdroj: vlastní aplikace



Obrázek 29: Happy: 20% redukce (217 542 plošek), Zdroj: vlastní aplikace

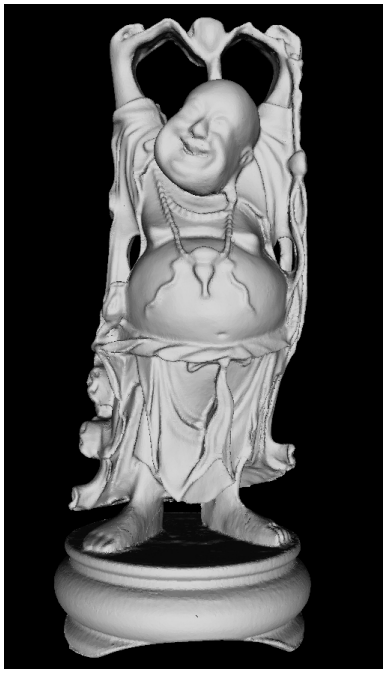


Obrázek 30: Happy: 50% redukce (Quadric error) (527 615 plošek), Zdroj: [15]

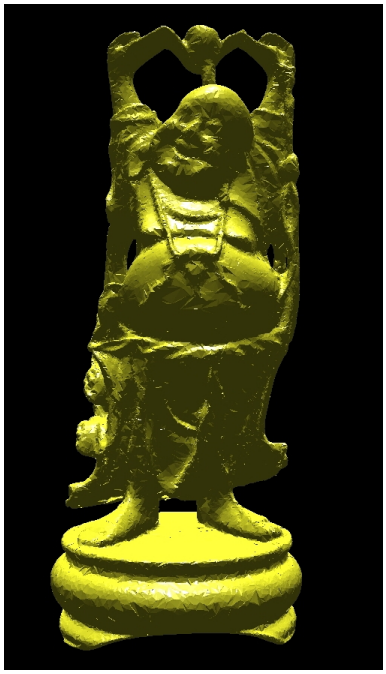


Obrázek 31: Happy: 20% redukce (Quadric error) (210 821 plošek), Zdroj: [15]

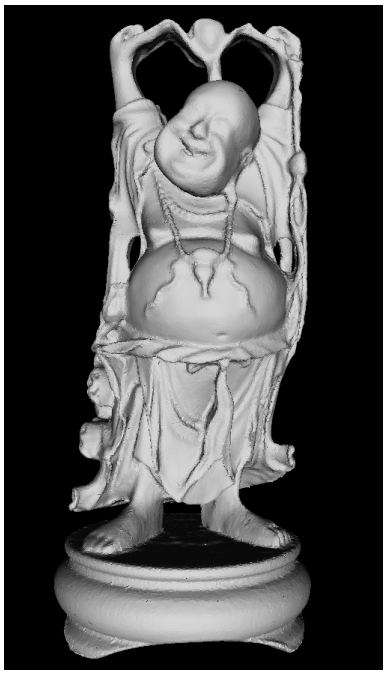
Obrázek 32: Happy: 50% redukce (543 858 plošek), Zdroj: vlastní aplikace



Obrázek 34: Happy: 50% redukce (Shortest edge) (539 204 plošek), Zdroj: [15]



Obrázek 33: Happy: 20% redukce (217 542 plošek), Zdroj: vlastní aplikace



Obrázek 35: Happy: 20% redukce (Shortest edge) (215 432 plošek), Zdroj: [15]

6 Závěr

V samotném úvodu této bakalářské práce byly popsány významné redukční metody, jejichž vliv na současný vývoj zjednodušovacích metod je stále značný i přes to, že některé z nich byly prezentovány před více než 10 lety.

Dále byla navržena aplikace, která implementovala jeden ze známých algoritmů pro redukci trojúhelníkových sítí. Daný algoritmus jsme pozměnili, za pomoci jemné úpravy metrické funkce (viz. kapitola 4.3.2), a tím se pokusili snížit výpočetní složitost. Daný algoritmus jsme úspěšně implementovali a zadanou aplikaci zprovoznili. Zmíněná aplikace je připojena ve formě multimediální přílohy této bakalářské práce na disku CD. V závěru práce jsme provedli testy, které nám pomohly určit kvalitu námi implementovaného algoritmu, a to jak z hlediska rychlosti, tak z hlediska kvality výsledných aproximací, které tento algoritmus produkuje.

6.1 Výhody a nedostatky

Jak již bylo řečeno výše, v kapitolách 5.2 a 5.3 jsme porovnali implementovaný algoritmus s různými redukčními metodami. Toto srovnání nám pomohlo aspoň přibližně zařadit náš algoritmus a stanovit jeho efektivitu, rychlost a kvalitu.

Námi implementovaný algoritmus se jeví jako efektivní a relativně rychlostně rovnocenný ve srovnání s algoritmem jemu podobným (viz. tabulka 4), jako je například algoritmus založený na Schroederově metodě prezentovaný v [12]. Rychlost našeho algoritmu je řádově stejná jako u algoritmů se srovnatelnou kvalitou aproximace.

Aproximace generované naší aplikací jsou v porovnání s aproximacemi, které byly vytvořeny pomocí aplikace $Mesh\ Simplification\ Viewer$ [15] méně kvalitní. Jelikož jsou implementované metody v aplikaci [15] založené na decimaci hran, jejich vyšší kvalita se dala očekávat.

Algoritmus byl implementován tak, aby při redukci dané sítě nevznikaly žádné nové hraniční či komplexní vrcholy. Tato vlastnost je bezesporu výhodou, vede však k nejpodstatnějšímu nedostatku námi prezentovaného algoritmu.

Jelikož při triangulaci vzniklých děr může dojít k anomálii (vznik hraničního či komplexního vrcholu), kterou nelze předem předvídat, jsme nuceni při jejím zjištění zrušit daný redukční krok. Tento proces je samozřejmě jednak časově náročnější a také snižuje počet vrcholů určených k redukci. Z tohoto důvodu nejsme například schopni vyprodukovat minimální aproximace daných modelů, jako jsou například aproximace obsahující pouhé 1% z původního počtu trojúhelníků.

6.2 Další vývoj

Po dokončení této práce je algoritmus i celá aplikace, která tento redukční algoritmus prezentuje, plně funkční. Při návrhu a implementaci aplikace však nebyly brány na zřetel jisté okolnosti použití algoritmu, které by jeho současný stav vylepšily. Jedná se ze zásady o problém jednosměrné redukce. Náš algoritmus byl implementován bez potřeby uchovávání informací o předchozích krocích redukce pro snížení paměťové náročnosti. Tato vlastnost však brání rychlému zpětnému redukování sítě. V případech, kdy je například model redukován na 20%, jeho zpětná redukce na 40% vyžaduje nastavení sítě do výchozího stavu (tedy originální síť) a poté redukování této sítě na požadovanou hodnotu. Tento jev by bylo do budoucna vhodné odstranit, pro realističtější simulaci práce s 3D modely.

7 Literatura

[1] Brodsky, D., Watson, B. *Model Simplification Through Refinement*. Dept. of Computer Science University of British Columbia, Dept. of Computing Science University of Alberta, 2000.

[2] Schroeder, W., Zarge, J., Lorensen, W. *Decimation of Triangle Meshes*. General Electric Company Schenectady, NY, 1992.

[3] Hoppe, H. *Progressive Meshes*. Microsoft Research, 1996.

[4] Hoppe, H., DeRose, T., Duchamp, T., McDonald, J., Stuetzle, W. *Mesh optimization*. Computer Graphics (SIGGRAPH '93 Proceedings) (1993), 19–26.

[5] Knapp, M. *Mesh Decimation Using VTK*. Institute of Computer Graphics and Algorithms, Vienna University of Technology, 2002.

[6] Garland, M., Heckbert, P. *Surface simplification using quadric error metrics*. SIGGRAPH 97 Conference Proceedings, 1997.

[7] Garland, M. *Multiresolution Modeling: Survey & Future Opportunities*. SIGGRAPH 97 course notes, 1997.

[8] Garland, M. *Quadric-based polygonal surface simplification*. The PhD thesis, 1999.

[9] Lindstrom, P., Turk, G. *Fast and memory efficient polygonal simplification*. IEEE Visualization 98 Conference Proceedings, 1998.

[10] Low, K., Tan, T. *Model simplification using vertex-clustering*. 1997 Symposium on Interactive 3D Graphics. ACM SIGGRAPH, 1997.

[11] Váša, L. *Methods for dynamic mesh size reduction - State of the Art and concept of doctoral thesis*. Department of Computer Science and Engineering, University of West Bohemia, 2006.

[12] Franc, M. *Metody redukce trojúhelníkových sítí, Diplomová práce*. Katedra informatiky a výpočetní techniky, Západočeská univerzita v Plzni, 2000.

[13] *Server University of Princeton* [online].
http://www.cs.princeton.edu/gfx/proj/sugcon/models/

[14] *Metody redukce trojúhelníkových sítí* [online].
http://herakles.zcu.cz/ marty/html/thesis/thesis.html

[15] *Jeff Sommer's Mesh Simplification Viewer* [online].
http://jsomers.com/vipm_demo/meshsimp.html